EE2703 : Applied Programming Lab
Assignment 7
Circuit Analysis using SymPy

Sai Shashank GP
EE20B040

April 1, 2022

0.1 Introduction

In this assignment, we are going to analyse some circuits of low and high pass filters using python simplification modules like **SymPy** and the previous concepts we have use in A6, like **scipy.signal** etc...
We have been given a LPF as example and solution to solve it using the mentioned libraries. We are going to solve for HPF similarly.

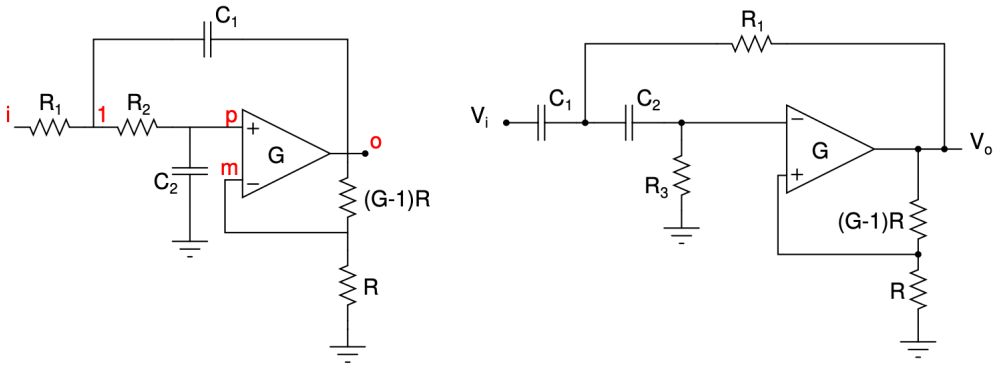

Figure 1: LPF and HPF circuits

0.2 Assignment

0.2.1 Prerequisites

We need to first import all the modules mentioned.

```
from pylab import *
import numpy as np
import scipy.signal as sp
import sympy as sm
```

and also the functions we might be using.

```
def vi(t):
    '''Returns the input voltage V_i(t) in Q2'''
    return np.where(t<0, 0, np.sin(2e3*np.pi*t) + np.cos(2e6*np.pi*t)),
                np.where(t<0, 0, np.sin(2e3*np.pi*t))

def u(t):
    '''Returns the unit step function'''
    return np.where(t<0, 0, 1)

def ds(t, w:float=1.5, a:float=0.05):
    return np.where(t<0, 0, np.sin(w*t)*np.exp(-1*a*t))

def symExpToLTI(exp, s=sm.symbols('s')):
    '''Returns the scipy lti object'''
```

```
    exp_num, exp_den = exp.as_numer_denom()
    num_coeffs = sm.Poly(exp_num, s).all_coeffs()
    den_coeffs = sm.Poly(exp_den, s).all_coeffs()
    num = poly1d([float(n) for n in num_coeffs])
    den = poly1d([float(n) for n in den_coeffs])
    return sp.lti(num, den)

def LPF(Vi=1, R1:float=10e3, R2:float=10e3, C1:float=1e-9, C2:float=1e-9, G:float=1.586)
    '''Gives the Transfer function H(s) of given LPF in assignment'''

    s = sm.symbols('s')

    A = sm.Matrix([[(1/R1)+(1/R2)+(s*C1), 0, (-1/R2), (-s*C1)],
     [(-1/R2), 0, (1/R2)+(s*C2), 0], [0, G, 0, -1], [0, -G, G, -1]])
    B = sm.Matrix([Vi/R1, 0, 0, 0])

    V = A.inv()*B
    Vo = V[3]
    # print(Vo)

    ww = np.logspace(0, 10, 1001)
    ss = 1j*ww
    hf = sm.lambdify(s, Vo, 'numpy')
    v = hf(ss)

    loglog(ww, abs(v), lw=2)
    title(r'$\mid H(j{\omega}\mid$ of given LPF')
    xlabel(r'$\omega (log) \longrightarrow$')
    ylabel(r'${\mid H(j{\omega}\mid} (dB) \longrightarrow$')
    grid(True)
    show()

    H = symExpToLTI(Vo)

    return H

def HPF(Vi=1, R1:float=10e3, R2:float=10e3, C1:float=1e-9, C2:float=1e-9, G:float=1.586)
    '''Gives the Transfer function H(s) of given HPF in assignment'''

    s = sm.symbols('s')

    A = sm.Matrix([[0, G, 0, -1], [0, -G, G, -1], [(-s*C2), 0, (s*C2)+(1/R2), 0],
    [(s*(C1+C2)) + (1/R1), 0, (-s*C2), (-1/R1)]])
    B = sm.Matrix([0, 0, 0, Vi*s*C1])

    V = A.inv()*B
    global Vo
    Vo = V[3]
```

```
    # print(Vo)

    ww = np.logspace(0, 10, 1001)
    ss = 1j*ww
    hf = sm.lambdify(s, Vo, 'numpy')
    v = hf(ss)

    loglog(ww, abs(v), lw=2)
    title(r'$\mid H(j{\omega}\mid$ of given HPF')
    xlabel(r'$\omega (log) \longrightarrow$')
    ylabel(r'${\mid H(j{\omega}\mid} (dB) \longrightarrow$')
    grid(True)
    show()

    H = symExpToLTI(Vo)

    return H
```

0.2.2 Q1

In this question, we are asked to calculated the step response of the LPF given. The LPF's transfer function is given as follows.

$$\boxed{|H(j\omega)| = \frac{2.52 \times 10^{-8}}{3.13 \times 10^{-18}s^2 + 7 \times 10^{-13}s + 3.13 \times 10^{-8}}} \tag{1}$$

This is an LPF and it can be shown using the magnitude bode plot of the transfer function

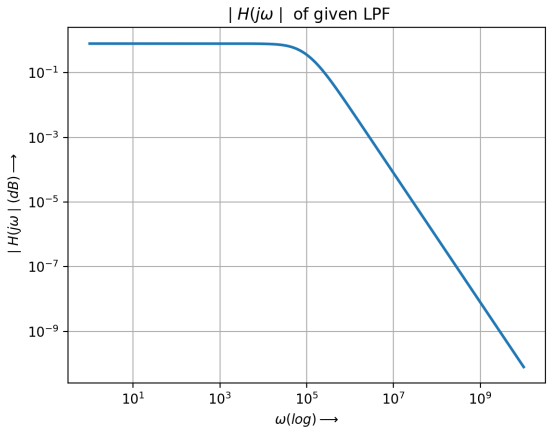

Figure 2: $|H(j\omega)|$ (dB) vs ω (log)

To calculate the step response of the LPF, we need to extract the transfer function first and then we can use **sp.lsim** to get the step response.

```
def Q1(t=np.linspace(0, 1e-3, 1001)):
    '''This function solves Q1'''
    H = LPF()
```

```
u_ = u(t)

_, u_resp, _ = sp.lsim(H, u_, t)

plot(t, u_resp)
title('Step response of given LPF')
xlabel(r'time $\longrightarrow$')
show()
```

The step response of the LPF in time domain looks like this.

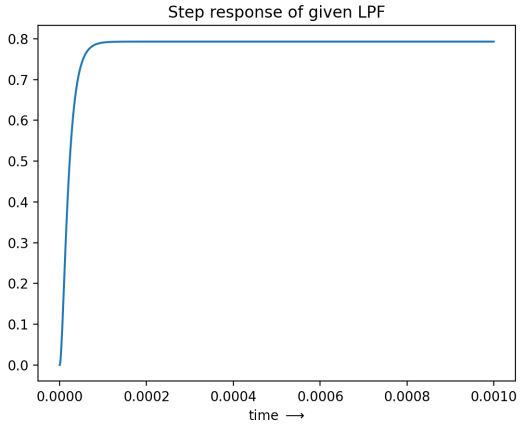

Figure 3: Step Response of given LPF

0.2.3 Observation

We can clearly see that the step function, though scaled down, is almost reconstructed even after passing through an LPF. It indicates that step function consists of frequency components which are $< 10^5$ rad/s.

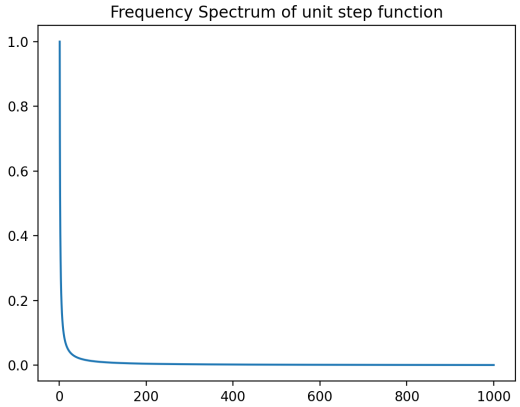

Figure 4: Frequency Spectrum of Step Function

This figure justifies the argument that step function is made of lower frequencies.

0.2.4 Q2

In this question, we are asked to find the output of LPF when we pass a signal with two frequencies in it. The signal we are passing is,

$$V_i(t) = (\sin(2000\pi t) + \cos(2 \times 10^6 t))u_0(t) \tag{2}$$

And calculation is done similar to the Q1, but instead of step input, we give this signal as input. So,

```
def Q2(t=np.linspace(0, 10e-3, 1000001)):
    '''This function solves Q2'''
    Vo = LPF()
    inp, lfc = vi(t)

    _, resp, _ = sp.lsim(Vo, inp, t)

    plot(t, resp, label='Vo')
    plot(t, lfc, label='LFC(Vi)')
    title('LPF on mixed frequencies')
    xlabel(r'time $\longrightarrow$')
    legend(loc='upper right')
    show()
```

The following is the output and along with it is the low frequency component(LFC).

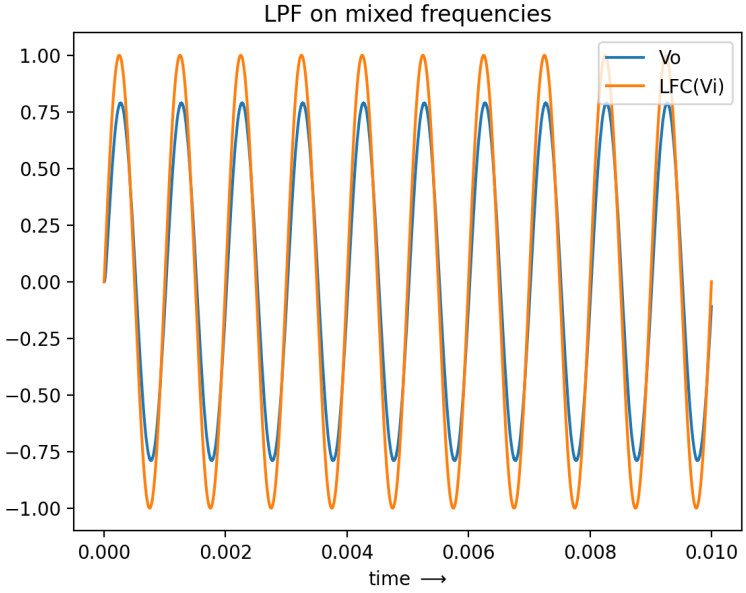

Figure 5: V_o(t) and LFC(V_i(t)) in time domain

We can see that the output is almost looking like a scaled down version of the low frequency component of input. So, we can see that the LPF filtered out the high frequency component of the input and gave out only low frequency component.

0.2.5 Q3

In this question, we are given a "High Pass Filter" circuit and we need to find out it's characteristics. And after solving the equation matrix in sympy. We are going to get the following approximately,

$$\boxed{|H(j\omega)| = \frac{2.52 \times 10^{-8} s^2}{3.13 \times 10^{-18} s^2 + 7 \times 10^{-13} s + 3.13 \times 10^{-8}}} \tag{3}$$

This is a HPF and it can be verified by plotting the magnitude bode plot of the transfer function.

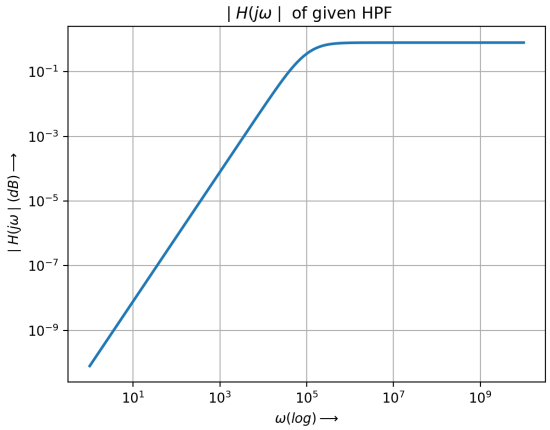

Figure 6: $|H(j\omega)|$ (dB) vs ω (log)

0.2.6 Observation

The transfer function's of LPF and HPF given are looking like in such a way that their sum is almost constant and it will pass all frequencies. And their multiplication looks like it pass no frequencies. Let us calculate and see how they are actually.

```
L = LPF()
H = HPF()

F1 = L * H
F2 =  L + H

f1 = symExpToLTI(F1)
f2 = symExpToLTI(F2)

w1, S1, phi1= f1.bode()
w2, S2, phi2= f2.bode()

semilogx(w1, S1)
semilogx(w2, S2)
grid(True)
show()
```

And the plots looks like this. As we can see, except for frequencies around 10^5 rad/s, it acts

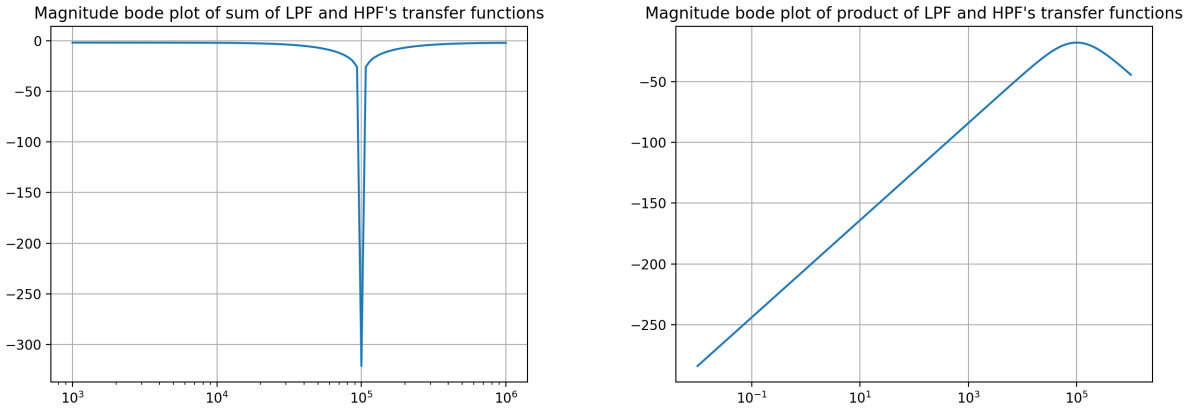

Figure 7: $H_l + H_h$ and $H_l.H_h$

as we discussed. The specialty of this observation is that it can be used to well understand the importance of LPF and HPF's transfer functions and how a small change in circuitry can affect the output. As we can see, the addition leads to an **All Pass Filter** almost and it can be realised by superposing both the filters. Multiplying leads to a **No Pass Filter** which can be realised by cascading both the filters.

0.2.7 Q4

In this question, we are asked to find the output of HPF when a damped sinusoid is passed into it. It is calculated as follows

```
def Q4(t=np.linspace(1e-3, 30, 10001)):
    '''This function solves Q4'''
    H = HPF()
    inp= ds(t)

    _, resp, _ = sp.lsim(H, inp, t)

    plot(t, resp, label='Vo')
    plot(t, inp, label='Vi')
    title('HPF on a damped sinusoid with low frequency')
    xlabel(r'time $\longrightarrow$')
    legend(loc='upper right')
    show()
```

By providing an appropriate time interval and frequency, we can see the following outputs.

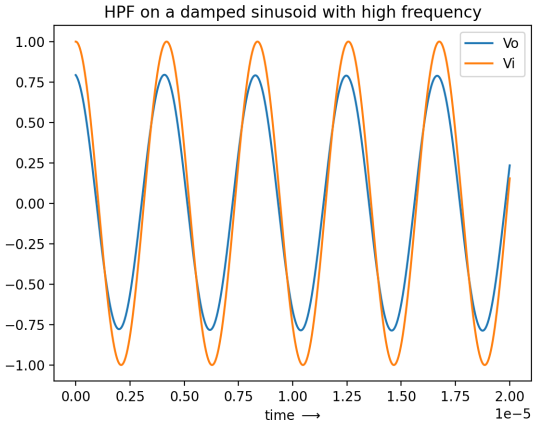

Figure 8: Output of HPF for a high frequency damped sinusoid respectively

When higher frequency is given, it gives an output which looks like a scaled down version of the actual input. But when lower frequency is given, it doesn't let it pass and it just gives no output.

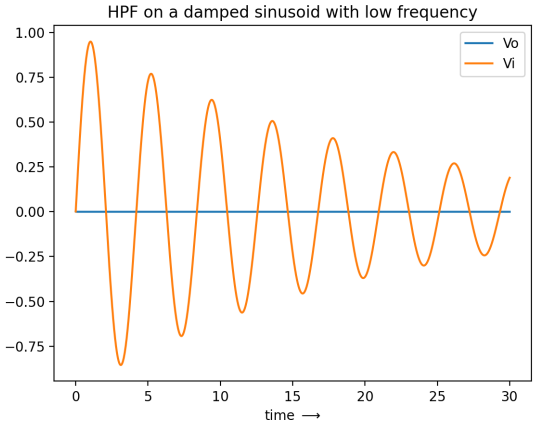

Figure 9: Outputs of HPF for a low frequency damped sinusoid respectively

0.2.8 Q5

In this question, we are asked to plot the step response of the given HPF. We can calculate and plot it as follows.

```
def Q5(t=np.linspace(1e-8, 1e-3, 10001)):
    '''This function solves Q5'''
    H = HPF()
    u_ = u(t)

    _, u_resp, _ = sp.lsim(H, u_, t)

    plot(t, u_resp)
    title('Step response of given HPF')
    xlabel(r'time $\longrightarrow$')
    show()
```

And the plot looks like the following.

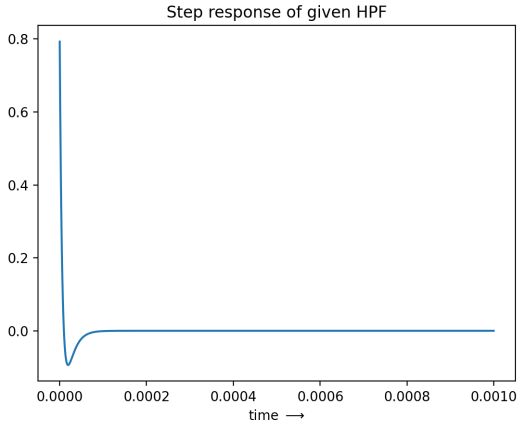

Figure 10: Step Response of given HPF

It is as we have imagined, the step response is high initially because of the sudden shooting in the step function. As we have observed in Q1 that step function is majorly composed of low frequencies, the step response of HPF dies out quickly.

0.3 TakeAways

- I have learnt how to partially automate the circuit analysis using SymPy's simplification classes
- I have gotten some clarity on how filters work and how can we realise them.
- I have understood how important sampling parameter is.
- I have understood how to correlate the circuits and it's transfer functions, how we can infer more from them than from the circuit itself.